Project 1-Report

DS8004-Data Mining

Author: Rafik Matta

Contents

Project Description

In this project, a custom implementation of Logistic Regression and Back-propagation for Multilayer Perceptron's (MLP) is being applied to two datasets from the UCI Machine Learning Repository.

The first dataset (Data Set 1) is a Mice Protein Expression Dataset which has 8 unique classes and 77 variables. The second dataset (Data Set 2) is a Forest Type Dataset which has 4 unique classes and 27 variables. With this information in mind, exploratory data analysis (EDA) followed by modeling and analysis was done to determine if applying the two above mentioned techniques can assist in properly classifying the data based on selected attributes.

Solution Method and Experiment Design

Implementation of Logistic Regression

The implementation of Logistic Regression used to perform analysis is based on stochastic gradient descent and updates the weights of the hyperplane in an online fashion (Alpaydin, 2014). It is based on the following:

$$y_i^t = \frac{\exp \boldsymbol{w}_i^T \boldsymbol{x}^t}{\sum_k \exp \boldsymbol{w}_k^T \boldsymbol{x}^t}$$

Equation 1. Output equation for multiclass Logistic Regression *(Alpaydin, 2014)*.

During testing, we calculate all y_k, k = 1, . . . , K and choose C_i if $y_i = \max_k y_k$. We use a cross entropy error measure as seen in Equation 2 to derive our update rule for gradient descent as seen in Equation 3 (Alpaydin, 2014).

$$E(\{\boldsymbol{w}_i, w_{i0}\}_i | \mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t$$

Equation 2. Cross Entropy Error measure.

$$\Delta w_{ij}^t = \eta (r_i^t - y_i^t) x_j^t$$

Equation 3. Gradient Descent update rule for the weights.

Using the online version of logistic regression allows the algorithm to converge much faster than the batch version, but ends up being more susceptible to changes in the input or might end up taking just as long or longer to converge if the samples aren't randomly ordered. There are slight differences and convergences from the referenced sources in the implementation which are further discussed in the "Common Enhancements and Considerations" section.

Implementation of MLP with Backpropagation

The implementation of MLP more closely follows the pseudo-code provided by the I2ML 3e text by AlPaydin in Figure 11.11 (Alpaydin, 2014). The backpropagation algorithm runs several epochs to reach convergence. Each epoch is split into feed forward and feedback sections.

<u>Feed forward steps</u>

1) Calculation of hidden unit values as seen in Equation 4 (Alpaydin, 2014).

$$z_h = \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^d w_{hj} x_j + w_{h0}\right)\right]}, \quad h = 1, \ldots, H$$

Equation 4. Calculation of hidden unit values with a sigmoid activation function

2) Calculation of output values prior to applying softmax for class discrimination (Alpaydin, 2014).

$$o_i^t = \sum_{h=1}^H v_{ih} z_h^t + v_{i0}$$

Equation 5. Calculation of output values with hidden weights.

3) Applying softmax to get a normalized value for log discrimination (Alpaydin, 2014).

$$y_i^t = \frac{\exp o_i^t}{\sum_k \exp o_k^t}$$

Equation 6. Calculation of class outputs using softmax.

Feed Back steps

The feedback requires us to measure error and determine update rules (Alpaydin, 2014).

$$E(\mathbf{W}, \mathbf{V}|X) = -\sum_t \sum_i r_i^t \log y_i^t$$

Equation 7. Cross Entropy Error Measure used to derive gradient-descent update rules.

The below update rules in Equation 8 are applied with each sample in an online fashion (Alpaydin, 2014).

$$\begin{aligned} \Delta v_{ih} &= \eta \sum_t (r_i^t - y_i^t) z_h^t \\ \Delta w_{hj} &= \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t \end{aligned}$$

Equation 8. Update Rules for input and hidden weights.

Common Enhancements and Considerations

One of the key differences or divergences that will be noticeable from the implementation as seen in the Appendix and the pseudo code is the use of an exp-trick used for numerical stability (Vieria, n.d.):

$$\pi_i = \frac{\exp(x_i - b)\exp(b)}{\sum_{j=1}^n \exp(x_j - b)\exp(b)} = \frac{\exp(x_i - b)}{\sum_{j=1}^n \exp(x_j - b)}$$

Equation 9. Exp-trick to bound the values within a range that a typical computer can handle.

The reason this was done is due to the lack of ability for a computer to handle values over a certain size (generally speaking floats can't be greater than 2^32 in precision).

Other enhancements that were considered but not applied is weight decay or a momentum measure (Alpaydin, 2014).

Normalization was also applied to the input data as seen in Equation 10 and will be elaborated on in the Experiment Design Section.

$$\frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$$

Equation 10. Standardization Calculation

Experiment Design

For each data set the following steps were taken:

1) Prepare Data – this involved separating out the variables chosen from the EDA phase, and hot coding the class variable to be used in training and testing
2) Shuffle and Split Data – randomness is quite important for online methods, and to guarantee that the data was random it was shuffled and split into 80% of samples for training and 20% for testing.

3) Normalize data – an additional step of normalization was done to ensure the variables were all standardized to values between 0-1 which improves convergence performance significantly and makes interpretability of the weights much easier
4) Apply Logistic Regression for 10,50,100 epochs to determine convergence point
5) Apply Back Propagation with 2 hidden units for 10,50,100,500 epochs to determine convergence point

Of note for this experiment, the step size was kept the same at 0.1 and hidden units were only increased if it was challenging to fit the model to the data.

Results

Exploratory Data Analysis

Data Set 1 – Mice Protein Expression

Link : (https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression)

Here is the description of the data from the UCI ML Repository site:

> "consists of the expression levels of 77 proteins/protein modifications that produced detectable signals in the nuclear fraction of cortex…The dataset contains a total of 1080 measurement per protein. Each measurement can be considered as an independent sample/mouse. The eight classes of mice are described based on features such as genotype, behavior and treatment. According to genotype, mice can be control or trisomic. According to behavior, some mice have been stimulated to learn (context-shock) and others have not (shock-context) and in order to assess the effect of the drug memantine in recovering the ability to learn in trisomic mice, some mice have been injected with the drug and others have not."

Based on the above description each sample can be treated independently and as the author of this report has no subject matter expertise in the domain of molecular biology, a naïve assumption was made that all the protein expressions can be treated as independent variables. As such the first 5 proteins were chosen:

- DYRK1A_N
- ITSN1_N
- BDNF_N
- NR1_N
- NR2A_N

In Table 1, we find the summary statistics of these variables.

	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N
count	1077.000000	1077.000000	1077.000000	1077.000000	1077.000000
mean	0.425810	0.617102	0.319088	2.297269	3.843934
std	0.249362	0.251640	0.049383	0.347293	0.933100
min	0.145327	0.245359	0.115181	1.330831	1.737540
25%	0.288121	0.473361	0.287444	2.057411	3.155678
50%	0.366378	0.565782	0.316564	2.296546	3.760855
75%	0.487711	0.698032	0.348197	2.528481	4.440011
max	2.516367	2.602662	0.497160	3.757641	8.482553

Table 1. Summary Statistics of chosen variables

From here, we can begin to explore the subset of chosen data. Figure 1 shows us the class distribution. Figure 2 is a combination of histograms for the variables as well as scatter plots of each combination of 2 variables to view possible linear dependence.

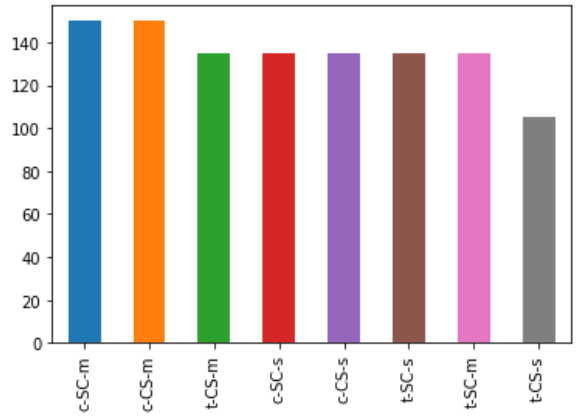

Figure 1. Histogram showing class distribution

Figure 1 shows us that distribution of classes in the data set is close to being uniform which should ensure that the trained models are not biased towards some classes over others. As can be seen in Figure 2, it looks like there's a strong linear correlation between some variables which is likely to impact results. Three of the variables have relatively normal (read Gaussian) distributions with few outliers based on their histograms, but the first two seem rather skewed to the left. Considering the models being applied are not probabilistic, this shouldn't have much impact.

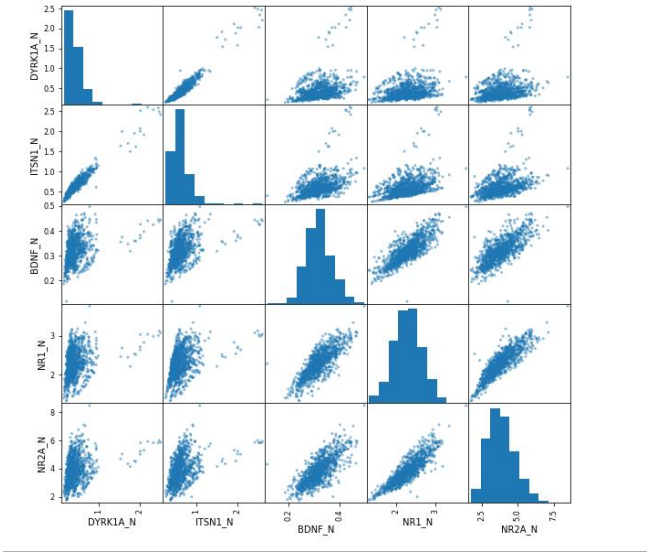

Figure 2. Histograms and scatter plots for chosen variables.

As can be seen in Figure 3, many of the variables are highly correlated, which can also be seen in the Figure 2 based on the scatter plots. Contrary to the initial assumption made, the lack of independence in the variables will likely affect the performance of the algorithms making convergence rather difficult.

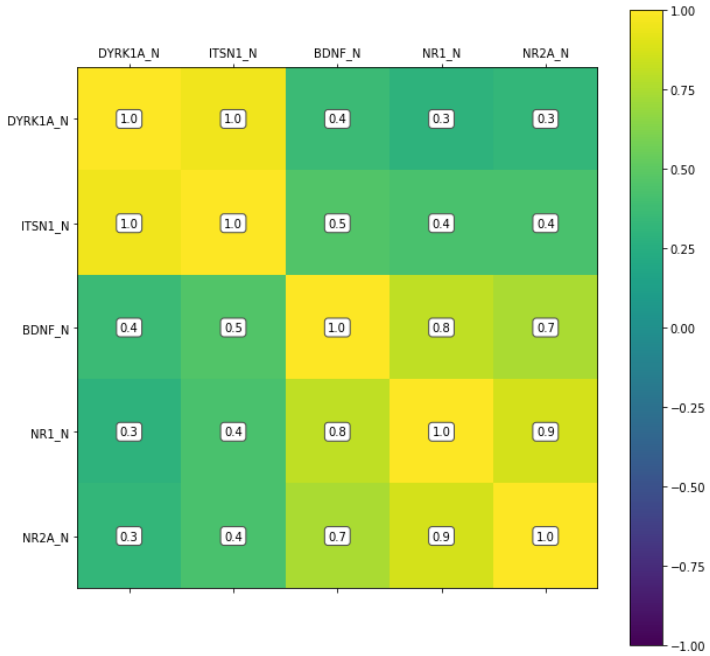

Figure 3. Correlation Matrix for chosen variables.

Data Set 2 – Forest Type Data

Link : (https://archive.ics.uci.edu/ml/datasets/Forest+type+mapping)

Here is the description of the data from the UCI ML Repository site:

> "contains training and testing data from a remote sensing study which mapped different forest types based on their spectral characteristics at visible-to-near infrared wavelengths, using ASTER satellite imagery. The output (forest type map) can be used to identify and/or quantify the ecosystem services (e.g. carbon storage, erosion protection) provided by the forest."

Based on the above description each sample can be treated independently and again, as the author of this report has no subject matter expertise in the domain of ecology or spectroscopy, a naïve assumption was made that there is likely to be high dependency amongst the variables given the homogeneity of the images of forests. Therefore, the goal of the exploration was to find independent variables. The following variables were chosen:

- b1
- b2
- pred_minus_obs_S_b1
- pred_minus_obs_S_b2
- pred_minus_obs_S_b9

Table 2 contains the summary statistics of the variables.

	b1	b2	pred_minus_obs_S_b1	pred_minus_obs_S_b2	pred_minus_obs_S_b9
count	523.000000	523.000000	523.000000	523.000000	523.000000
mean	59.887189	39.380497	-20.017304	-1.056195	-4.201377
std	12.345004	15.930120	3.806830	1.375642	1.518350
min	31.000000	23.000000	-32.950000	-8.800000	-10.830000
25%	51.500000	28.000000	-22.505000	-1.795000	-4.940000
50%	58.000000	32.000000	-19.990000	-1.030000	-4.130000
75%	67.000000	48.000000	-18.080000	-0.300000	-3.245000
max	107.000000	160.000000	5.130000	12.460000	7.790000

Table 2. Summary statistics of chosen variables

From here, we can begin to explore the subset of chosen data. Figure 4 shows us the class distribution. Figure 5 is again a combination of histograms for the variables as well as scatter plots of each combination of 2 variables to view possible linear dependence.

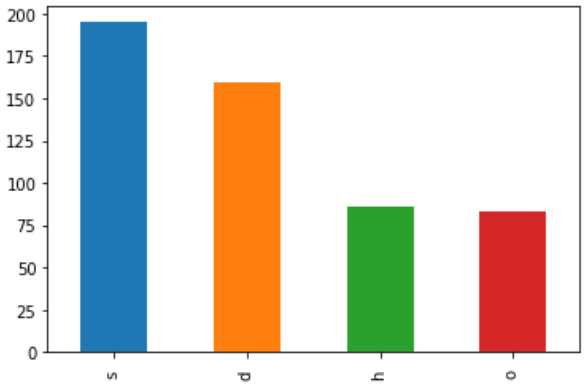

Figure 4. Class Distribution of the data set

Figure 4 shows us that distribution of classes in the data set is not quite uniform and there is a skew towards 's' class data. This might impact model training and introduce bias towards this class. As can be seen in Figure 5 there is no strong linear

dependence amongst the variables which should help in convergence. Four of the five variables have relatively normal (read Gaussian) distributions with few outliers based on their histograms.

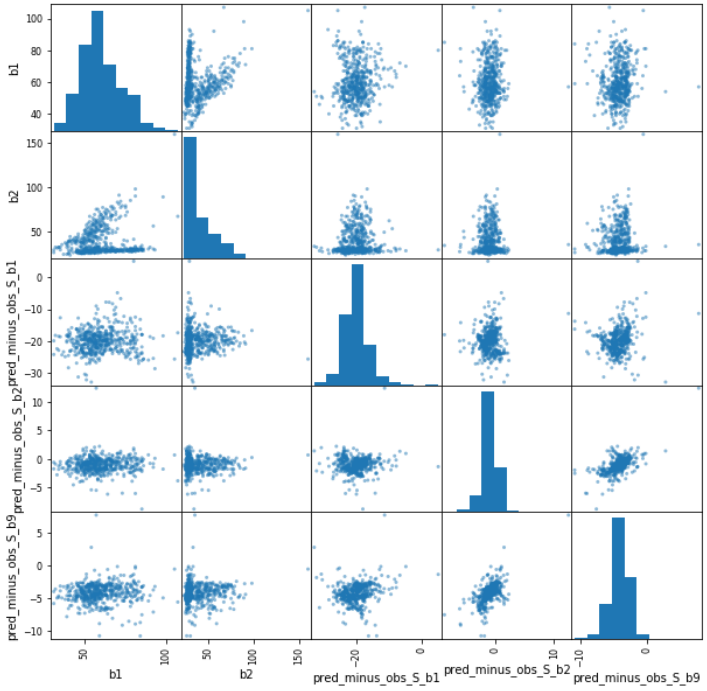

Figure 5. Histograms and scatter plots for chosen variables.

As can be seen in Figure 6, our variables have mostly week correlations which confirms the observations made in Figure 5. This should allow training the models to converge must faster thus improving validation set accuracy.

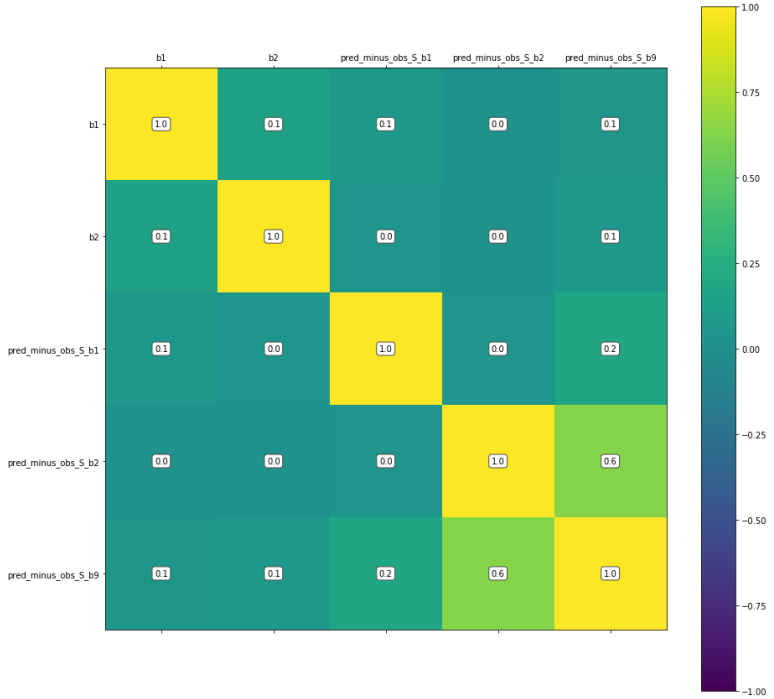

Figure 6. Correlation Matrix for chosen variables.

Analysis

Logistic Regression was applied to both datasets with a step size (η) of 0.1. Results can be seen in Table 3 and Table 5. Back Propagation was applied to both datasets with H=2 (two hidden units) and step size (η) equal to 0.1. Results are in Table 4 and Table 6. Accuracy was used as our measure of performance based on the following:

$$accuracy = \frac{\sum \#\ of\ correct\ predictions\ per\ class}{\sum \#\ of\ predictions\ made}$$

Increasing number of hidden units on both data sets did not significantly improve convergence and as such the results are omitted to keep the analysis brief and consistent. Increasing/decreasing step size also had no noticeable effect on convergence performance and as such results are omitted.

NUMBER OF EPOCHS	TRAINING ACCURACY (%)	TESTING ACCURACY (%)
10	30.66	32.87
50	37.86	35.65
100	39.49	37.5
500	45.41	44.91

Table 3. Results for Logistic Regression on Data Set 1

NUMBER OF EPOCHS	TRAINING ACCURACY (%)	TESTING ACCURACY (%)
10	16.26	18.98
50	36.94	36.57
100	36.24	36.11
500	40.53	43.05

Table 4. Results for Back Propagation for Data set 1

NUMBER OF EPOCHS	TRAINING ACCURACY (%)	TESTING ACCURACY (%)
10	78.47	68.57
50	82.30	79.05
100	82.30	79.05
500	83.25	79.05

Table 5. Results for Logistic Regression on Data Set 2

NUMBER OF EPOCHS	TRAINING ACCURACY (%)	TESTING ACCURACY (%)
10	41.87	39.05
50	83.01	76.19
100	83.73	76.19
500	85.65	78.09

Table 6. Results for Back Propagation on Data Set 2

Conclusion and Learnings

As can be seen from the analysis, the choice of variables and their dependence matters tremendously. For Data Set 1, it is clear the strong correlation between variables and linear dependence made convergence rather difficult. It can be observed that between 100 and 500 iterations of the algorithm that there's an improvement, albeit a small one indicating that both logistic regression and backpropagation would require much more training time to reach any significantly improved performance. On the other hand, Data Set 2 due to its highly independent variables, was able to converge quickly and discriminate much better between classes increasing overall accuracy. Another interesting thing to note is that it seems that on these data sets, logistic regression will perform better at smaller training intervals whereas back propagation only begins to match or surpass logistic regression with higher epochs.

References

Alpaydin, E. (2014). Introduction to Machine Learning, 3e.

Vieria, T. (n.d.). *Graduate Descent*. Retrieved from https://timvieira.github.io/blog/post/2014/02/11/exp-normalize-trick/

Appendix

Exploratory Data Analysis

```
import random
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
import plotly.plotly as py
import plotly.graph_objs as go

def plot_corr(df,size=10):
    '''Function plots a graphical correlation matrix for each pair of columns in the dataframe.

    Input:
        df: pandas DataFrame
        size: vertical and horizontal size of the plot'''

    corr = df.corr()
    fig, ax = plt.subplots(figsize=(size, size))
    plt.colorbar(ax.matshow(corr,vmin=-1,vmax=1))

    for (i, j), z in np.ndenumerate(corr):
        ax.text(j, i, '{:0.1f}'.format(z), ha='center', va='center',
            bbox=dict(boxstyle='round', facecolor='white', edgecolor='0.3'))

    plt.xticks(range(len(corr.columns)), corr.columns);
    plt.yticks(range(len(corr.columns)), corr.columns);
```

Data Set 1

```
data_set_1 = pd.read_csv('data_cortex_nuclear.csv')
data_set_1_sub = data_set_1[['DYRK1A_N','ITSN1_N','BDNF_N','NR1_N','NR2A_N','class']]
data_set_1_sub.describe()
```

```
data_set_1['class'].value_counts().plot(kind='bar')
```

```
plot_df = data_set_1[['DYRK1A_N','ITSN1_N','BDNF_N','NR1_N','NR2A_N']]
pd.scatter_matrix(plot_df, figsize=(10, 10), diagonal='hist');
```

```
plot_corr(data_set_1_sub)
```

Dataset 2

```
data_set_2_training = pd.read_csv('forest_type_training.csv')
data_set_2_testing = pd.read_csv('forest_type_testing.csv')
data_set_2_full = pd.concat([data_set_2_training,data_set_2_testing])
```

```
data_set_2_sub = data_set_2_full[['b1','b2','pred_minus_obs_S_b1','pred_minus_obs_S_b2','pred_minus_obs_S_b9','class']]
data_set_2_sub.describe()
```

```
data_set_2_sub['class'].value_counts().plot(kind='bar')
```

```
plot_df = data_set_2_sub[['b1','b2','pred_minus_obs_S_b1','pred_minus_obs_S_b2','pred_minus_obs_S_b9']]
pd.scatter_matrix(plot_df, figsize=(10, 10), diagonal='hist');
```

```
plot_corr(data_set_2_sub,size=15)
```

Data Preparation

Dataset 1

```
data_set_1_sub = data_set_1_sub.sample(frac=1)
data_set_1_sub = data_set_1_sub.dropna()
temp_matrix_1 = data_set_1_sub[['DYRK1A_N','ITSN1_N','BDNF_N','NR1_N','NR2A_N']].as_matrix()
data_input_matrix_1 = np.zeros(shape=(len(temp_matrix_1),temp_matrix_1.shape[1]+1))

for t in range(len(data_input_matrix_1)):
    data_input_matrix_1[t,0] = 1

for t in range(len(data_input_matrix_1)):
    for i in range(1,data_input_matrix_1.shape[1]):
```

```
        data_input_matrix_1[t,i] = temp_matrix_1[t,i-1]

classes = data_set_1_sub['class'].unique()

expected_matrix_1 = np.zeros(shape=(len(temp_matrix_1),len(classes)))

for t in range(len(expected_matrix_1)):
    index_val = np.where(classes==data_set_1_sub[['class']].iloc[t]['class'])[0][0]
    expected_matrix_1[t,index_val] = 1

data_input_matrix_1


expected_matrix_1
```

Dataset 2

```
temp_matrix_2 = data_set_2_sub[['b1','b2','pred_minus_obs_S_b1','pred_minus_obs_S_b2','pred_minus_obs_S_b9']].as_matrix()
data_input_matrix_2 = np.zeros(shape=(len(temp_matrix_2),temp_matrix_2.shape[1]+1))

for t in range(len(data_input_matrix_2)):
    data_input_matrix_2[t,0] = 1

for t in range(len(data_input_matrix_2)):
    for i in range(1,data_input_matrix_2.shape[1]):
        data_input_matrix_2[t,i] = temp_matrix_2[t,i-1]

classes = data_set_2_sub['class'].unique()

expected_matrix_2 = np.zeros(shape=(len(temp_matrix_2),len(classes)))

for t in range(len(expected_matrix_2)):
    index_val = np.where(classes==data_set_2_sub[['class']].iloc[t]['class'])[0][0]
    expected_matrix_2[t,index_val] = 1

data_input_matrix_2


expected_matrix_2
```

Algorithm Implementation

Logistic Regression

```
def log_regression(x,r,step_size,accuracy_stop,iterations):
    d = x.shape[1]
    K = r.shape[1]
    weights = np.zeros(shape=(K,d))

    for i in range(K):
        for j in range(d):
            weights[i,j] = random.uniform(-0.01, 0.01)

    for run in range(iterations):
        accuracy_score = 0

        for t in range(len(x)):
            o = []
            y = []

            #Feed forward
            for i in range(K):
               # o.append(0)
                #for j in range(d):
                o.append(np.dot(weights[i,].transpose(),x[t,]))

            max_o = max(o)
            o2 = [math.exp(x-max_o) for x in o]
            denom = sum(o2)

            for i in range(K):
                y.append(o2[i]/denom)

            if  r[t].argmax() == y.index(max(y)):
                accuracy_score = accuracy_score + 1

            #Update Equation for online
            for i in range(K):
                for j in range(d):
                     weights[i,j] = weights[i,j] + step_size*(r[t,i]-y[i])*x[t,j]

        if (accuracy_score/len(x) >= accuracy_stop):
            print("Training Accuracy: " + str(accuracy_score/len(x)*100))
            return weights

    print("Training Accuracy: " + str(accuracy_score/len(x)*100))
    return weights
```

```
def test_log_regression(x,r,weights):
    error_score = 0
    K = r.shape[1]
    d = x.shape[1]
    for t in range(len(x)):
        o = []
        y = []

        for i in range(K):
            o.append(np.dot(weights[i,].transpose(),x[t,]))

        max_o = max(o)
        o2 = [math.exp(x-max_o) for x in o]
        denom = sum(o2)

        for i in range(K):
            y.append(o2[i]/denom)

        if r[t].argmax() == y.index(max(y)):
            error_score = error_score + 1
    accuracy = (error_score/len(x))*100
    print("Test Accuracy: " + str(accuracy))
    return(accuracy)
```

Back Propogation 1-Layer

```
def back_prop(x,r,H,step_size,accuracy_stop,iterations):
    d = x.shape[1]
    K = r.shape[1]
    H = H+1
    v_weights = np.zeros(shape=(K,H))
    w_weights = np.zeros(shape=(H,d))


    for i in range(K):
        for h in range(H):
            v_weights[i,h] = random.uniform(-0.01, 0.01)

    for h in range(1,H):
        for j in range(d):
            w_weights[h,j] = random.uniform(-0.01, 0.01)

    for run in range(iterations):
        error_score = 0
        delta_v = np.zeros(shape=(K,H))
        delta_w = np.zeros(shape=(H,d))

        for t in range(len(x)):

            #feed forward of hidden units
            z = []
            z.append(1)

            for h in range(1,H):
                z.append(0)
                for j in range(d):
                    z[h] = z[h] + w_weights[h,j]*x[t,j]

            z_bounded = []
            z_bounded.append(1)
            for index in range(1,len(z)):
                if z[index] >=0:
                    z_bounded.append(1/(1+math.exp(-z[index] )))
                else:
                    z_bounded.append(math.exp(z[index])/(1+math.exp(z[index])))

            o = []
            y = []
            for i in range(K):
                o.append(0)
                for h in range(H):
                    o[i] = o[i] + v_weights[i,h]*z_bounded[h]

            #feed forward of outputs units
            max_o = max(o)
            o2 = [math.exp(x-max_o) for x in o]
            denom = sum(o2)

            for i in range(K):
                y.append(o2[i]/denom)

            if r[t].argmax() == y.index(max(y)):
                error_score = error_score + 1

            for i in range(K):
                for h in range(H):
                    delta_v[i,h] = step_size*(r[t,i]-y[i])*z_bounded[h]
```

```
            for h in range(1,H):
                diff = 0
                for i in range(K):
                    diff = diff + (r[t,i]-y[i])*v_weights[i,h]
                for j in range(d):
                    delta_w[h,j] = step_size*diff*z_bounded[h]*(1-z_bounded[h])*x[t,j]


            for i in range(K):
                for h in range(H):
                    v_weights[i,h] = v_weights[i,h] + delta_v[i,h]

            for h in range(1,H):
                for j in range(d):
                    w_weights[h,j] = w_weights[h,j] + delta_w[h,j]

        if (error_score/len(x) >= accuracy_stop):
            print("Training Accuracy: " + str(error_score/len(x)*100))
            return (v_weights,w_weights)

    print("Training Accuracy: " + str(error_score/len(x)*100))
    return (v_weights,w_weights)


def test_back_prop(x,r,H,v_weights,w_weights):
    error_score = 0
    H=H+1
    K = r.shape[1]
    d = x.shape[1]
    for t in range(len(x)):
        #print(t)
        #feed forward of hidden units
        z = []
        z.append(1)

        for h in range(1,H):
            z.append(0)
            for j in range(d):
                z[h] = z[h] + w_weights[h,j]*x[t,j]

        z_bounded = []
        z_bounded.append(1)
        for index in range(1,len(z)):
            if z[index] >=0:
                z_bounded.append(1/(1+math.exp(-z[index] )))
            else:
                z_bounded.append(math.exp(z[index])/(1+math.exp(z[index])))

        o = []
        y = []
        for i in range(K):
            o.append(0)
            for h in range(H):
                o[i] = o[i] + v_weights[i,h]*z_bounded[h]

        #feed forward of outputs units
        max_o = max(o)
        o2 = [math.exp(x-max_o) for x in o]
        denom = sum(o2)

        for i in range(K):
            y.append(o2[i]/denom)

        if r[t].argmax() == y.index(max(y)):
            error_score = error_score + 1

    accuracy = (error_score/len(x))*100
    print("Test Accuracy: " + str(accuracy))
    return(accuracy)
```

Testing the Algorithms with Mock Data

Generate the Data

```
data = np.zeros(shape=(9000,6))
x = np.zeros(shape=(9000,3)) #1,x1,x2
r = np.zeros(shape=(9000,3)) #class1,class2,class3

for i in range(1,3):
    for t in range(9000):
        if t%3 == 0:
            data[t,i] = random.gauss(25, 1)
            data[t,3] = 1
        elif t%2 == 0:
            data[t,i] = random.gauss(15, 1)
            data[t,4] = 1
        else:
            data[t,i] = random.gauss(5, 1)
            data[t,5] = 1
```

```
for t in range(9000):
    data[t,0] = 1

random.shuffle(data)

x = data[:,0:3]
r = data[:,3:6]

plt.scatter(x[:,1],x[:,2])
plt.show()
```

```
x_norm = np.zeros(shape=(len(x),x.shape[1]))
for i in range(1,x.shape[1]):
    for t in range(len(x)):
        x_norm[t,i] = (x[t,i]-min(x[:,i]))/(max(x[:,i])-min(x[:,i]))

for t in range(len(x)):
        x_norm[t,0] = 1
```

Test Log Regression

```
weights = log_regression(x_norm[:7000],r[:7000],0.1,0.7,10)
accuracy = test_log_regression(x_norm[7000:],r[7000:],weights)
```

Test Back Prop

```
H = 2
weights = back_prop(x_norm[:7000],r[:7000],H,0.1,0.7,1)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[7000:],r[7000:],H,v_weights,w_weights)
```

Training and Testing Data Set 1

```
x_norm = np.zeros(shape=(len(data_input_matrix_1),data_input_matrix_1.shape[1]))
for i in range(1,data_input_matrix_1.shape[1]):
    for t in range(len(data_input_matrix_1)):
        x_norm[t,i] = (data_input_matrix_1[t,i]-min(data_input_matrix_1[:,i]))/(max(data_input_matrix_1[:,i])-min(data_input_matrix_1[:,i]))

for t in range(len(data_input_matrix_1)):
        x_norm[t,0] = 1
```

Log Regression

10

```
weights = log_regression(x_norm[:861],expected_matrix_1[:861],0.1,0.7,10)
accuracy = test_log_regression(x_norm[861:],expected_matrix_1[861:],weights)
```

50

```
weights = log_regression(x_norm[:861],expected_matrix_1[:861],0.1,0.7,50)
accuracy = test_log_regression(x_norm[861:],expected_matrix_1[861:],weights)
```

100

```
weights = log_regression(x_norm[:861],expected_matrix_1[:861],0.1,0.7,100)
accuracy = test_log_regression(x_norm[861:],expected_matrix_1[861:],weights)
```

500

```
weights = log_regression(x_norm[:861],expected_matrix_1[:861],0.1,0.7,500)
accuracy = test_log_regression(x_norm[861:],expected_matrix_1[861:],weights)
```

Back Prop

H=2, 10

```
H = 2
weights = back_prop(x_norm[:861],expected_matrix_1,H,0.1,0.7,10)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[861:],expected_matrix_1[861:],H,v_weights,w_weights)
```

H=2, 50

```
H = 2
weights = back_prop(x_norm[:861],expected_matrix_1,H,0.1,0.7,50)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[861:],expected_matrix_1[861:],H,v_weights,w_weights)
```

H=2, 100

```
H = 10
weights = back_prop(x_norm[:861],expected_matrix_1,H,0.1,0.7,100)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[861:],expected_matrix_1[861:],H,v_weights,w_weights)
```

H=2, 500

```
H = 2
weights = back_prop(x_norm[:861],expected_matrix_1,H,0.1,0.7,500)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[861:],expected_matrix_1[861:],H,v_weights,w_weights)
```

Training and Testing Data Set 2

```
x_norm = np.zeros(shape=(len(data_input_matrix_2),data_input_matrix_2.shape[1]))
for i in range(1,data_input_matrix_2.shape[1]):
    for t in range(len(data_input_matrix_2)):
        x_norm[t,i] = (data_input_matrix_2[t,i]-min(data_input_matrix_2[:,i]))/(max(data_input_matrix_2[:,i])-min(data_input_matrix_2[:,i]))

for t in range(len(data_input_matrix_2)):
        x_norm[t,0] = 1
```

Log Regression

10

```
weights = log_regression(x_norm[:418],expected_matrix_2[:418],0.1,1,10)
accuracy = test_log_regression(x_norm[418:],expected_matrix_2[418:],weights)
```

50

```
weights = log_regression(x_norm[:418],expected_matrix_2,0.1,1,50)
accuracy = test_log_regression(x_norm[418:],expected_matrix_2[418:],weights)
```

100

```
weights = log_regression(x_norm[:418],expected_matrix_2,0.1,1,100)
accuracy = test_log_regression(x_norm[418:],expected_matrix_2[418:],weights)
```

500

```
weights = log_regression(x_norm[:418],expected_matrix_2,0.1,1,500)
accuracy = test_log_regression(x_norm[418:],expected_matrix_2[418:],weights)
```

Back Prop

H=2,10

```
H = 2
weights = back_prop(x_norm[:418],expected_matrix_2,H,0.1,1,10)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[418:],expected_matrix_2[418:],H,v_weights,w_weights)
```

H=2,50

```
H = 2
weights = back_prop(x_norm[:418],expected_matrix_2,H,0.1,1,50)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[418:],expected_matrix_2[418:],H,v_weights,w_weights)
```

H=2,100

```
H = 2
weights = back_prop(x_norm[:418],expected_matrix_2,H,0.1,1,100)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[418:],expected_matrix_2[418:],H,v_weights,w_weights)
```

H=2, 500

```
H = 2
weights = back_prop(x_norm[:418],expected_matrix_2,H,0.1,1,500)
v_weights = weights[0]
w_weights = weights[1]
accuracy = test_back_prop(x_norm[418:],expected_matrix_2[418:],H,v_weights,w_weights)
```